Contents

External Authorization

7 minute read ✔ page test

This task shows you how to set up an Istio authorization policy using a new experimental value for the action field, `CUSTOM`, to delegate the access control to an external authorization system. This can be used to integrate with OPA authorization, `oauth2-`

`proxy`, your own custom external authorization server and more.

The following information describes an experimental feature, which is intended for evaluation purposes only.

Before you begin

Before you begin this task, do the following:

- Read the Istio authorization concepts.
- Follow the Istio installation guide to install Istio.
- Deploy test workloads:

  This task uses two workloads, `httpbin` and `sleep`, both deployed in namespace `foo`. Both workloads run with an Envoy proxy sidecar. Deploy the `foo` namespace and workloads with the following command:

```
$ kubectl create ns foo
$ kubectl label ns foo istio-injection=enabled
$ kubectl apply -f @samples/httpbin/httpbin.yaml@ -n foo
$ kubectl apply -f @samples/sleep/sleep.yaml@ -n foo
```

- Verify that `sleep` can access `httpbin` with the following command:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl ht
tp://httpbin.foo:8000/ip -s -o /dev/null -w "%{http_code}\n
"
200
```

If you don’t see the expected output as you follow the task, retry after a few seconds.

Caching and propagation overhead can cause some delay.

Deploy the external authorizer

First, you need to deploy the external authorizer. For this, you will simply deploy the sample external authorizer in a standalone pod in the mesh.

1. Run the following command to deploy the sample external authorizer:

```
$ kubectl apply -n foo -f https://raw.githubusercontent.com
/istio/istio/release-1.11/samples/extauthz/ext-authz.yaml
service/ext-authz created
deployment.apps/ext-authz created
```

2. Verify the sample external authorizer is up and running:

```
$ kubectl logs "$(kubectl get pod -l app=ext-authz -n foo -
o jsonpath={.items..metadata.name})" -n foo -c ext-authz
2021/01/07 22:55:47 Starting HTTP server at [::]:8000
2021/01/07 22:55:47 Starting gRPC server at [::]:9000
```

Alternatively, you can also deploy the external

authorizer as a separate container in the same pod of the application that needs the external authorization or even deploy it outside of the mesh. In either case, you will also need to create a service entry resource to register the service to the mesh and make sure it is accessible to the proxy.

The following is an example service entry for an external authorizer deployed in a separate container in the same pod of the application that needs the external authorization.

```
apiVersion: networking.istio.io/v1alpha3
kind: ServiceEntry
metadata:
  name: external-authz-grpc-local
spec:
  hosts:
  - "external-authz-grpc.local" # The service name to be used in
 the extension provider in the mesh config.
  endpoints:
  - address: "127.0.0.1"
  ports:
  - name: grpc
    number: 9191 # The port number to be used in the extension p
rovider in the mesh config.
    protocol: GRPC
  resolution: STATIC
```

Define the external authorizer

In order to use the `CUSTOM` action in the authorization policy, you must then define the external authorizer that is allowed to be used in the mesh. This is currently defined in the `extension provider` in the mesh config.

Currently, the only supported extension provider type is the `Envoy ext_authz` provider. The external authorizer must implement the corresponding Envoy `ext_authz`

check API.

In this task, you will use a sample external authorizer which allows requests with the header `x-ext-authz: allow`.

1. Edit the mesh config with the following command:

   ```
   $ kubectl edit configmap istio -n istio-system
   ```

2. In the editor, add the extension provider definitions shown below:

   The following content defines two external providers `sample-ext-authz-grpc` and `sample-ext-`

`authz-http` using the same service `ext-authz.foo.svc.cluster.local`. The service implements both the HTTP and gRPC check API as defined by the Envoy `ext_authz` filter. You will deploy the service in the following step.

```
data:
  mesh: |-
    # Add the following content to define the external auth
orizers.
    extensionProviders:
    - name: "sample-ext-authz-grpc"
      envoyExtAuthzGrpc:
        service: "ext-authz.foo.svc.cluster.local"
        port: "9000"
    - name: "sample-ext-authz-http"
      envoyExtAuthzHttp:
        service: "ext-authz.foo.svc.cluster.local"
        port: "8000"
        includeHeadersInCheck: ["x-ext-authz"]
```

Alternatively, you can modify the extension provider to control the behavior of the `ext_authz` filter for things like what headers to send to the

external authorizer, what headers to send to the application backend, the status to return on error and more. For example, the following defines an extension provider that can be used with the `oauth2-proxy`:

```
data:
  mesh: |-
    extensionProviders:
    - name: "oauth2-proxy"
      envoyExtAuthzHttp:
        service: "oauth2-proxy.foo.svc.cluster.local"
        port: "4180" # The default port used by oauth2-prox
y.
        includeHeadersInCheck: ["authorization", "cookie"]
# headers sent to the oauth2-proxy in the check request.
        headersToUpstreamOnAllow: ["authorization", "path",
 "x-auth-request-user", "x-auth-request-email", "x-auth-req
uest-access-token"] # headers sent to backend application w
hen request is allowed.
        headersToDownstreamOnDeny: ["content-type", "set-co
okie"] # headers sent back to the client when request is de
nied.
```

3. Restart Istiod to allow the change to take effect

with the following command:

```
$ kubectl rollout restart deployment/istiod -n istio-system
deployment.apps/istiod restarted
```

Enable with external authorization

The external authorizer is now ready to be used by the authorization policy.

1. Enable the external authorization with the following command:

   The following command applies an authorization policy with the `CUSTOM` action value for the `httpbin` workload. The policy enables the external authorization for requests to path `/headers` using the external authorizer defined by `sample-ext-authz-grpc`.

```
$ kubectl apply -n foo -f - <<EOF
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: ext-authz
spec:
  selector:
```

```
    matchLabels:
      app: httpbin
  action: CUSTOM
  provider:
    # The provider name must match the extension provider d
efined in the mesh config.
    # You can also replace this with sample-ext-authz-http
to test the other external authorizer definition.
    name: sample-ext-authz-grpc
  rules:
  # The rules specify when to trigger the external authoriz
er.
  - to:
    - operation:
        paths: ["/headers"]
EOF
```

At runtime, requests to path `/headers` of the `httpbin` workload will be paused by the `ext_authz` filter,

and a check request will be sent to the external authorizer to decide whether the request should be allowed or denied.

2. Verify a request to path `/headers` with header `x-ext-authz: deny` is denied by the sample `ext_authz` server:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -H "x-ext-authz: deny" -s
denied by ext_authz for not found header `x-ext-authz: allo
w` in the request
```

3. Verify a request to path `/headers` with header `x-ext-authz: allow` is allowed by the sample `ext_authz`

server:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/headers" -H "x-ext-authz: allow" -s
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin:8000",
    "User-Agent": "curl/7.76.0-DEV",
    "X-B3-Parentspanid": "430f770aeb7ef215",
    "X-B3-Sampled": "0",
    "X-B3-Spanid": "60ff95c5acdf5288",
    "X-B3-Traceid": "fba72bb5765daf5a430f770aeb7ef215",
    "X-Envoy-Attempt-Count": "1",
    "X-Ext-Authz": "allow",
    "X-Ext-Authz-Check-Result": "allowed",
    "X-Forwarded-Client-Cert": "By=spiffe://cluster.local/n
s/foo/sa/httpbin;Hash=e5178ee79066bfbafb1d98044fcd0cf80db76
be8714c7a4b630c7922df520bf2;Subject=\"\";URI=spiffe://clust
```

```
er.local/ns/foo/sa/sleep"
  }
}
```

4. Verify a request to path `/ip` is allowed and does not trigger the external authorization:

```
$ kubectl exec "$(kubectl get pod -l app=sleep -n foo -o js
onpath={.items..metadata.name})" -c sleep -n foo -- curl "h
ttp://httpbin.foo:8000/ip" -s -o /dev/null -w "%{http_code}
\n"
200
```

5. Check the log of the sample `ext_authz` server to confirm it was called twice (for the two requests). The first one was allowed and the second one was denied:

```
$ kubectl logs "$(kubectl get pod -l app=ext-authz -n foo -
o jsonpath={.items..metadata.name})" -n foo -c ext-authz
2021/01/07 22:55:47 Starting HTTP server at [::]:8000
2021/01/07 22:55:47 Starting gRPC server at [::]:9000
2021/01/08 03:25:00 [gRPCv3][denied]: httpbin.foo:8000/head
ers, attributes: source:{address:{socket_address:{address:"
10.44.0.22"  port_value:52088}}  principal:"spiffe://cluste
r.local/ns/foo/sa/sleep"}  destination:{address:{socket_add
ress:{address:"10.44.3.30"  port_value:80}}  principal:"spi
ffe://cluster.local/ns/foo/sa/httpbin"}  request:{time:{sec
onds:1610076306  nanos:473835000}  http:{id:"1386914285578 3
664817"  method:"GET"  headers:{key:":authority"  value:"ht
tpbin.foo:8000"}  headers:{key:":method"  value:"GET"}  hea
ders:{key:":path"  value:"/headers"}  headers:{key:"accept"
  value:"*/*"}  headers:{key:"content-length"  value:"0"}
headers:{key:"user-agent"  value:"curl/7.74.0-DEV"}  header
s:{key:"x-b3-sampled"  value:"1"}  headers:{key:"x-b3-spani
d"  value:"377ba0cdc2334270"}  headers:{key:"x-b3-traceid"
 value:"635187cb20d92f62377ba0cdc2334270"}  headers:{key:"x
-envoy-attempt-count"  value:"1"}  headers:{key:"x-ext-auth
```

z" value:"deny"} headers:{key:"x-forwarded-client-cert" value:"By=spiffe://cluster.local/ns/foo/sa/httpbin;Hash=dd14782fa2f439724d271dbed846ef843ff40d3932b615da650d028db655fc8d;Subject=\"\";URI=spiffe://cluster.local/ns/foo/sa/sleep"} headers:{key:"x-forwarded-proto" value:"http"} headers:{key:"x-request-id" value:"9609691a-4e9b-9545-ac71-3889bc2dffb0"} path:"/headers" host:"httpbin.foo:8000" protocol:"HTTP/1.1"}} metadata_context:{}
2021/01/08 03:25:06 [gRPCv3][allowed]: httpbin.foo:8000/headers, attributes: source:{address:{socket_address:{address:"10.44.0.22" port_value:52184}} principal:"spiffe://cluster.local/ns/foo/sa/sleep"} destination:{address:{socket_address:{address:"10.44.3.30" port_value:80}} principal:"spiffe://cluster.local/ns/foo/sa/httpbin"} request:{time:{seconds:1610076300 nanos:925912000} http:{id:"17959492964333813435" method:"GET" headers:{key:":authority" value:"httpbin.foo:8000"} headers:{key:":method" value:"GET"} headers:{key:":path" value:"/headers"} headers:{key:"accept" value:"*/*"} headers:{key:"content-length" value:"0"} headers:{key:"user-agent" value:"curl/7.74.0-DEV"} heade

rs:{key:"x-b3-sampled" value:"1"} headers:{key:"x-b3-span
id" value:"a66b5470e922fa80"} headers:{key:"x-b3-traceid"
value:"300c2f2b90a618c8a66b5470e922fa80"} headers:{key:"
x-envoy-attempt-count" value:"1"} headers:{key:"x-ext-aut
hz" value:"allow"} headers:{key:"x-forwarded-client-cert"
value:"By=spiffe://cluster.local/ns/foo/sa/httpbin;Hash=d
d14782fa2f439724d271dbed846ef843ff40d3932b615da650d028db655
fc8d;Subject=\"\";URI=spiffe://cluster.local/ns/foo/sa/slee
p"} headers:{key:"x-forwarded-proto" value:"http"} heade
rs:{key:"x-request-id" value:"2b62daf1-00b9-97d9-91b8-ba61
94ef58a4"} path:"/headers" host:"httpbin.foo:8000" proto
col:"HTTP/1.1"}} metadata_context:{}

You can also tell from the log that mTLS is enabled for the connection between the `ext-authz` filter and the sample `ext-authz` server because the source principal is populated with the value `spiffe://cluster.local/ns/foo/sa/sleep`.

You can now apply another authorization policy for the sample `ext-authz` server to control who is allowed to access it.

Clean up

1. Remove the namespace `foo` from your configuration:

   ```
   $ kubectl delete namespace foo
   ```

2. Remove the extension provider definition from

the mesh config.